Laporan Tugas Besar 1 IF3270 Pembelajaran Mesin

Bagian B

Anggota :

13521010	Muhamad Salman Hakim Alfarisi
13521013	Eunice Sarah Siregar
13521018	Syarifa Dwi Purnamasari
13521027	Agsha Athalla Nurkareem

Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2023

Daftar Isi

Penjelasan Implementasi

Neural Network merupakan sebuah metode kecerdasan buatan yang mengajarkan komputer untuk memproses data dengan cara yang terinspirasi oleh otak manusia. Dalam *machine learning*, *neural network* mengacu pada sekumpulan algoritma yang di *design* untuk membantu mesin dalam menemukan pattern tanpa diprogram secara eksplisit. Dalam neural network terdapat tiga hal dasar, yaitu *input layer*, *hidden layer*, dan *output layer*.

Pada kali ini, dirancang algoritma untuk melakukan *backpropagation* dari modul *feed forward* yang menerima input yang dalam bentuk file JSON dengan memanfaatkan beberapa *library* yang terdapat pada Python, seperti numpy untuk melakukan perhitungan, json untuk load model dari file JSON. Berikut adalah turunan fungsi aktivasi dan fungsi loss yang diimplementasikan.

Untuk turunan ReLU jika $x < 0$ maka $\frac{d}{dx}ReLU(x) = 0$, jika $x \geq 0$ $\frac{d}{dx}ReLU(x) = 1$.

Untuk turunan Sigmoid $\frac{d}{dx}\sigma(x) = \sigma(x) \times (1 - \sigma(x))$.

Untuk turunan Linear $\frac{d}{dx}x = 1$.

Untuk turunan Softmax jika $j \neq target$ maka $\frac{\partial E_d}{\partial net(x)} = p_j$, jika $j = target$ maka $\frac{\partial E_d}{\partial net(x)} = -(1 - p_j)$.

Untuk ReLU, sigmoid, dan linear memiliki fungsi loss $E = \frac{1}{2}\sum_{k \epsilon output}(t_k - o_k)^2$.

Untuk Softmax memiliki fungsi loss $E = -log(p_k),\ k = target.$

Proses Implementasi dimulai dari fungsi pembacaan file csv dan json pada fungsi readCSV akan menghasilkan output `id_list, sepal_length_list, sepal_width_list, petal_length_list, petal_width_list, species_list.` Fungsi readJSON akan menghasilkan output `input_size, layer, weights, input_array, target, learning_rate, batch_size, max_iteration, error_threshold, num_layer.`

```
class fileSystem:
   @staticmethod
   def readCSV(filename):
       id_list = []
       sepal_length_list = []
       sepal_width_list = []
       petal_length_list = []
       petal_width_list = []
       species_list = []
       with open(filename + '.csv', 'r') as csv_file:
           csv_reader = csv.reader(csv_file)
           next(csv_reader)
           for row in csv_reader:
               id = row[0]
               sepal_length = row[1].split(',')
               sepal_width = row[2].split(',')
               petal_length = row[3].split(',')
               petal_width = row[4].split(',')
               species = row[5].split(',')

               id_list.append(id)
               sepal_length_list.append(sepal_length)
               sepal_width_list.append(sepal_width)
               petal_length_list.append(petal_length)
               petal_width_list.append(petal_width)
               species_list.append(species)

       return id_list, sepal_length_list, sepal_width_list, petal_length_list,
petal_width_list, species_list

   def readJSON(filename):
       with open("testcaseB/" + filename + ".json", 'r') as file:
           data = json.load(file)
           input_size = data["case"]["model"]["input_size"]
           layer = data["case"]["model"]["layers"]
           weights = data["case"]["initial_weights"]
           input_array = np.array(data["case"]["input"])
           target = np.array(data["case"]["target"])
           learning_rate = data["case"]["learning_parameters"]["learning_rate"]
           batch_size = data["case"]["learning_parameters"]["batch_size"]
           max_iteration = data["case"]["learning_parameters"]["max_iteration"]
           error_threshold = data["case"]["learning_parameters"]["error_threshold"]
           num_layer = len(layer)

       return input_size, layer, weights, input_array, target, learning_rate, batch_size,
max_iteration, error_threshold, num_layer
```

Selanjutnya dibuat kelas forwardActivation untuk membuat fungsi aktivasi forward.

```
class forwardActivation:
   def relu(x):
       return np.maximum(0,x)
   def sigmoid(x):
       return 1 / (1 + np.exp(-x))
   def linear(x):
       return x
   def softmax(x):
       return np.exp(x) / np.sum(np.exp(x))
```

Selanjutnya dibuat kelas backActivationOutput untuk membuat fungsi aktivasi back pada output layer.

```
class backActivationOutput:
   def relu(x):
       if x >= 0:
           return 1
       else:
           return 0
   def linear(x):
       return 1
   def sigmoid(x):
       return x*(1 - x)
   def softmax(x):
       return 0
```

Selanjutnya dibuat kelas backActivationHiddem untuk membuat fungsi aktivasi back pada hidden layer.

```
class backActivationHidden:
    def relu(x, y):
        if x >= 0:
            return -(x-y)
        else:
            return 0
    def linear(x, y):
        return -(x-y)
    def sigmoid(x, y):
        return -y * (1-y) * (x-y)
    def softmax(a, b, x: int, y: int):
        return b if (x != y) else (-1 * (1 - a))
```

Selanjutnya dibuat kelas Layer untuk inisialisasi layer.

```
class Layer:
   def __init__(self, neurons, activation, weights=None, bias=1):
       self.neurons = neurons
       self.activation_name = activation
       self.activation_func = forwardActivation.__dict__[activation]
       self.activation_derivative_output = backActivationOutput.__dict__[activation]
       self.activation_derivative_hidden = backActivationHidden.__dict__[activation]
       self.weights = weights
       self.bias = bias
       self.outputs = None  # Initialize outputs attribute to None

   def set_outputs(self, outputs):
       self.outputs = outputs
```

Selanjutnya terdapat kelas NeuralNetwork, yang berisi add_layer untuk penambahan layer, forward_pass untuk forward propagation, calculate_loss untuk menghitung loss, backward_pass untuk backward propagation, train untuk training dataset, addLayer untuk perbandingan, save_model untuk menyimpan model ke dalam pickle, dan load_model untuk load model dari pickle.

```
class NeuralNetwork:
    def __init__(self, learning_rate=0.1, batch_size=10, max_iterations=100, 
error_threshold=0.1):
        self.layers = []
        self.learning_rate = learning_rate
        self.batch_size = batch_size
        self.max_iteration = max_iterations
        self.error_threshold = error_threshold

    def add_layer(self, num_neurons, activation_function, weights, bias):
        self.layers.append(Layer(
            neurons=num_neurons,
            activation=activation_function,
            weights=weights,
            bias=bias,
        ))

    def forward_pass(self, input_array):
        outputs = input_array
        for i, layer in enumerate(self.layers):
            layer_output = layer.activation_func(np.dot(outputs, layer.weights) + layer.bias)
            layer.set_outputs(layer_output)  # Set outputs attribute
            outputs = layer_output
        return outputs

    def calculate_loss(self, outputs, targets):
        return np.mean(np.square(targets - outputs))

    def backward_pass(self, input_array, targets, outputs):
        delta = -(targets - outputs) * self.layers[-1].activation_derivative_output(outputs)
        for i in range(len(self.layers) - 1, -1, -1):
            layer = self.layers[i]

            if i != 0:
                delta_weights = np.dot(self.layers[i - 1].outputs.T, delta)
            else:
                delta_weights = np.dot(input_array.T, delta)
            layer.weights -= self.learning_rate * delta_weights

            delta_bias = np.sum(delta, axis=0)
            layer.bias -= self.learning_rate * delta_bias

            if i > 0:
                delta = np.dot(delta, self.layers[i].weights.T) *
self.layers[i].activation_derivative_hidden(self.layers[i - 1].outputs, outputs)

    def train(self, input_array, targets):
        for epoch in range(self.max_iteration):
            for i in range(0, len(input_array), self.batch_size):
                input_batch = input_array[i:i + self.batch_size]
                target_batch = targets[i:i + self.batch_size]

                # Forward pass for the entire batch
                outputs = self.forward_pass(input_batch)

                # Backward pass for the entire batch
                self.backward_pass(input_batch, target_batch, outputs)

                # Calculate loss for the entire batch
                loss = self.calculate_loss(outputs, target_batch)

                print(f"Iteration {epoch + 1}, Loss: {loss}")

                # Check for early stopping
                if loss < self.error_threshold:
                    print("Training converged: Error threshold reached.")
                    return
            if epoch == self.max_iteration - 1:
                print("Training stopped by max iteration.")

        print("Final Weights:")
        for i, layer in enumerate(self.layers):
            print(f"Layer {i+1}:")
            print(layer.bias)
            print(layer.weights)

    def addLayer(self, layer: Layer) -> None:
        if self.layers:
            layer.input_shape = self.layers[-1].output_shape
        self.layers.append(layer)

    def save_model(self, filename):
        with open(filename, 'wb') as f:
            pickle.dump(self, f)

    def load_model(filename):
        with open(filename, 'rb') as f:
            return pickle.load(f)
```

```
iris = load_iris()
X_iris = iris.data
y_iris = iris.target

X_train, X_val, y_train, y_val = train_test_split(X_iris, y_iris, test_size=0.3,
random_state=42)

model_iris_sklearn = MLPClassifier(hidden_layer_sizes=(4,), activation="logistic",
learning_rate_init=0.1,
                                   batch_size=10, max_iter=1000, solver="sgd", random_state=42)
model_iris_sklearn.fit(X_train, y_train)
predictions_sklearn = model_iris_sklearn.predict(X_val)

report_sklearn = classification_report(y_val, predictions_sklearn)

mlp_custom = NeuralNetwork(learning_rate=0.1, max_iterations=1000, batch_size=10,
error_threshold=0.01)
mlp_custom.addLayer(Layer(input_shape=X_train.shape[1], neurons=4, activation='sigmoid'))
mlp_custom.addLayer(Layer(neurons=3, activation='linear'))  # Output layer
```

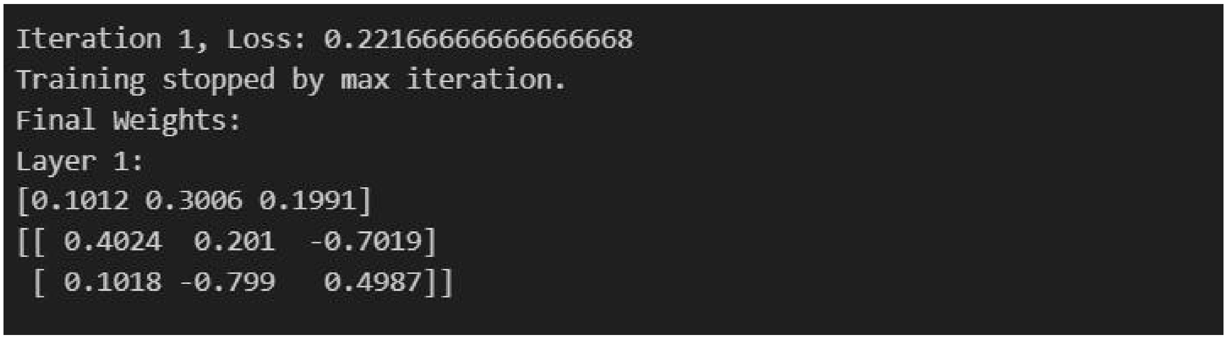
```
y_train_onehot = np.eye(3)[y_train]
stop_reason = mlp_custom.fit(X_train, y_train_onehot)

predictions_custom = np.argmax(mlp_custom.predict(X_val), axis=1)

report_custom = classification_report(y_val, predictions_custom)
```

Hasil Pengujian

Setelah melakukan perancangan algoritma, dilakukan pengujian untuk setiap hasil test case yang telah diberikan. Berikut adalah hasil pengujian dari setiap test case.

1. Linear

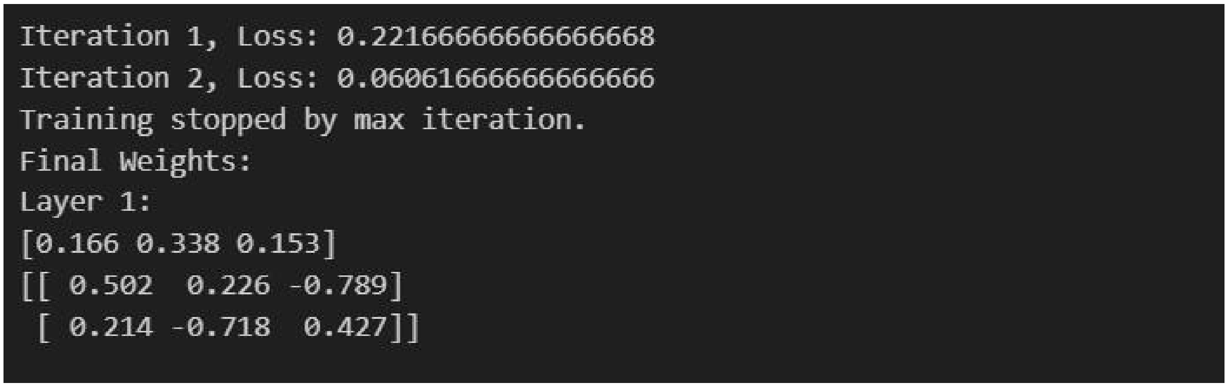
Hasil Pengujian



```
Iteration 1, Loss: 0.22166666666666668
Training stopped by max iteration.
Final Weights:
Layer 1:
[0.1012 0.3006 0.1991]
[[ 0.4024  0.201  -0.7019]
 [ 0.1018 -0.799   0.4987]]
```

2. Linear Two Iteration

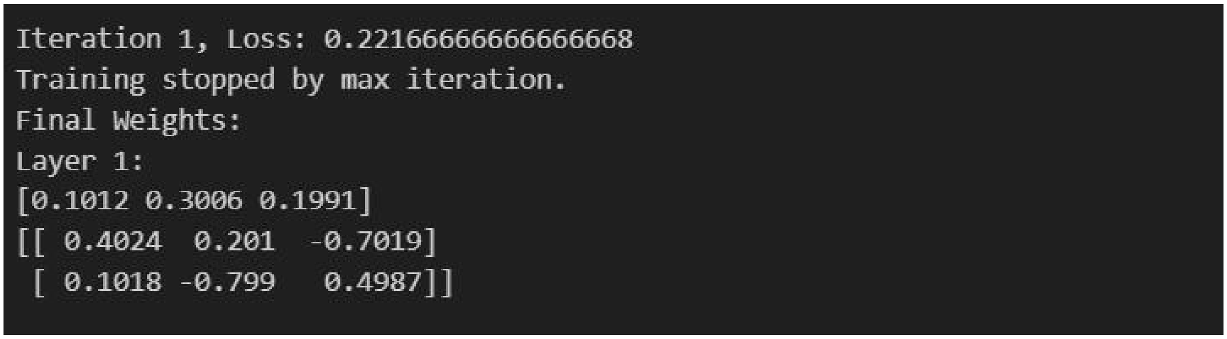
Hasil Pengujian



```
Iteration 1, Loss: 0.22166666666666668
Iteration 2, Loss: 0.06061666666666666
Training stopped by max iteration.
Final Weights:
Layer 1:
[0.166 0.338 0.153]
[[ 0.502  0.226 -0.789]
 [ 0.214 -0.718  0.427]]
```

3. Linear Small LR

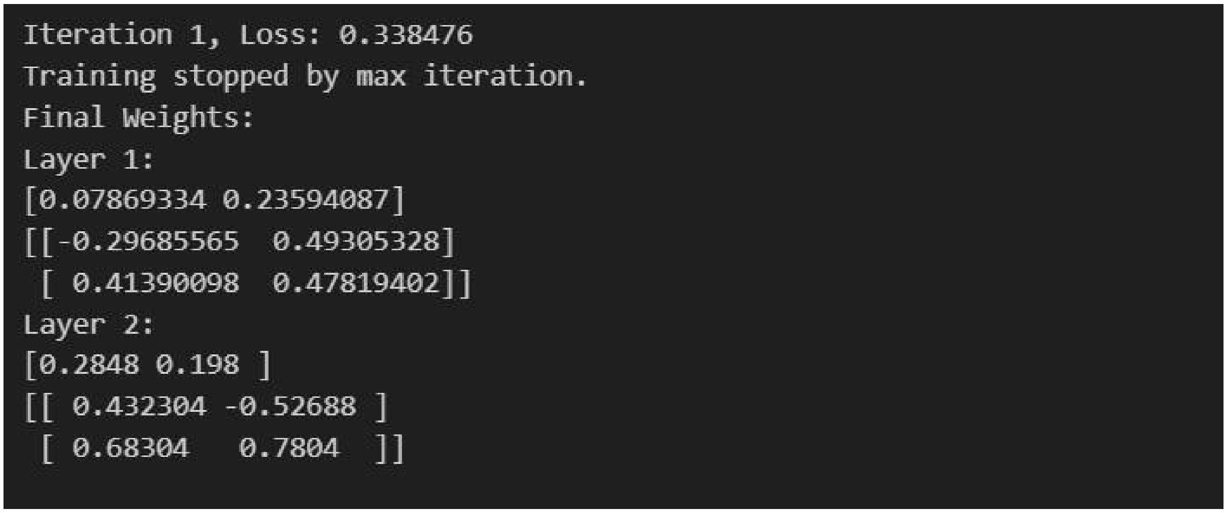
Hasil Pengujian



```
Iteration 1, Loss: 0.22166666666666668
Training stopped by max iteration.
Final Weights:
Layer 1:
[0.1012 0.3006 0.1991]
[[ 0.4024  0.201  -0.7019]
 [ 0.1018 -0.799   0.4987]]
```

4. MLP

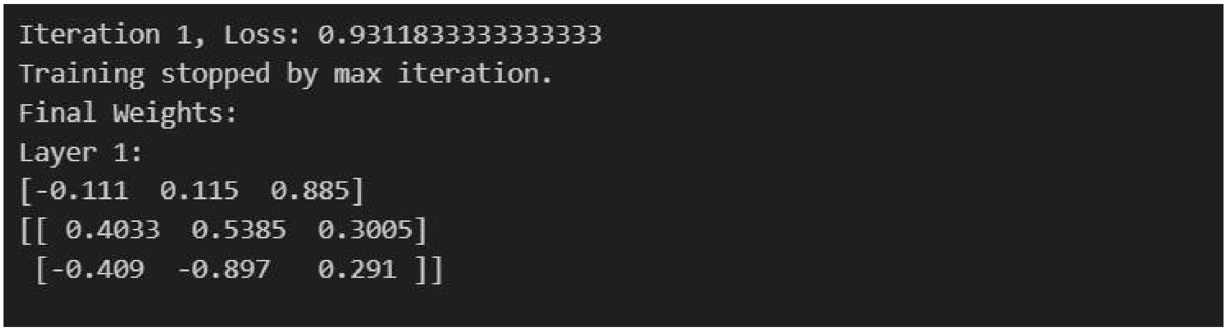
Hasil Pengujian



```
Iteration 1, Loss: 0.338476
Training stopped by max iteration.
Final Weights:
Layer 1:
[0.07869334 0.23594087]
[[-0.29685565  0.49305328]
 [ 0.41390098  0.47819402]]
Layer 2:
[0.2848 0.198 ]
[[ 0.432304 -0.52688 ]
 [ 0.68304   0.7804  ]]
```

5. Relu B

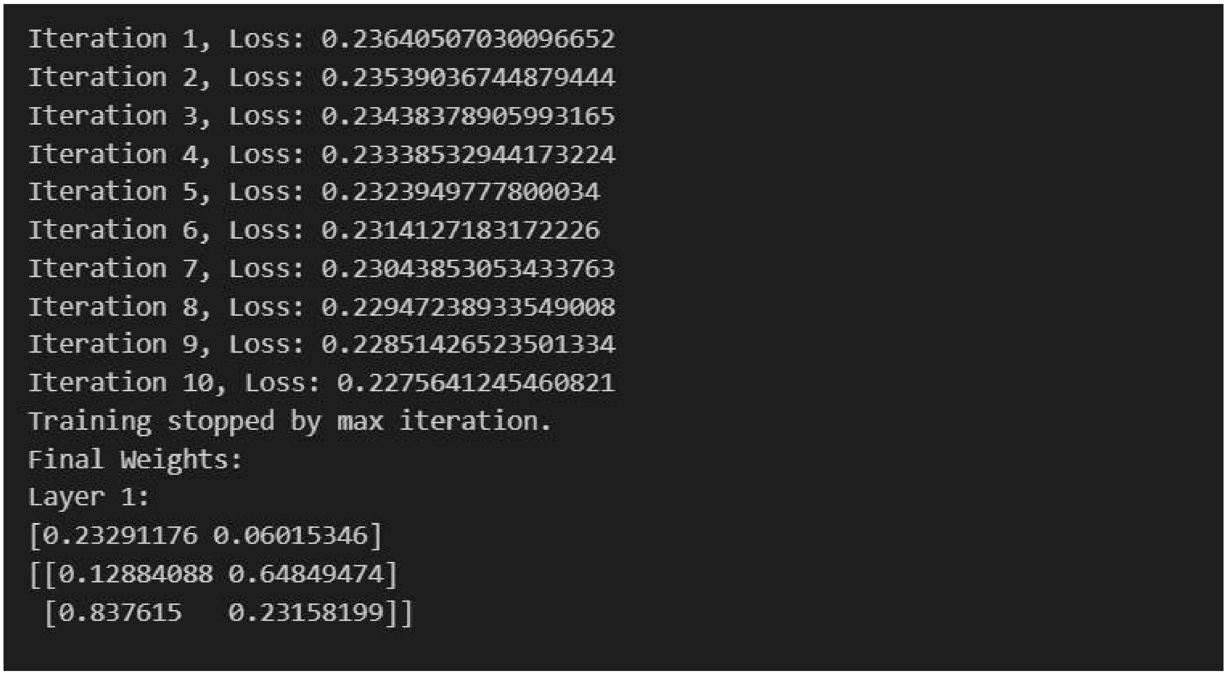
Hasil Pengujian



```
Iteration 1, Loss: 0.9311833333333333
Training stopped by max iteration.
Final Weights:
Layer 1:
[-0.111  0.115  0.885]
[[ 0.4033  0.5385  0.3005]
 [-0.409  -0.897   0.291 ]]
```

6. Sigmoid

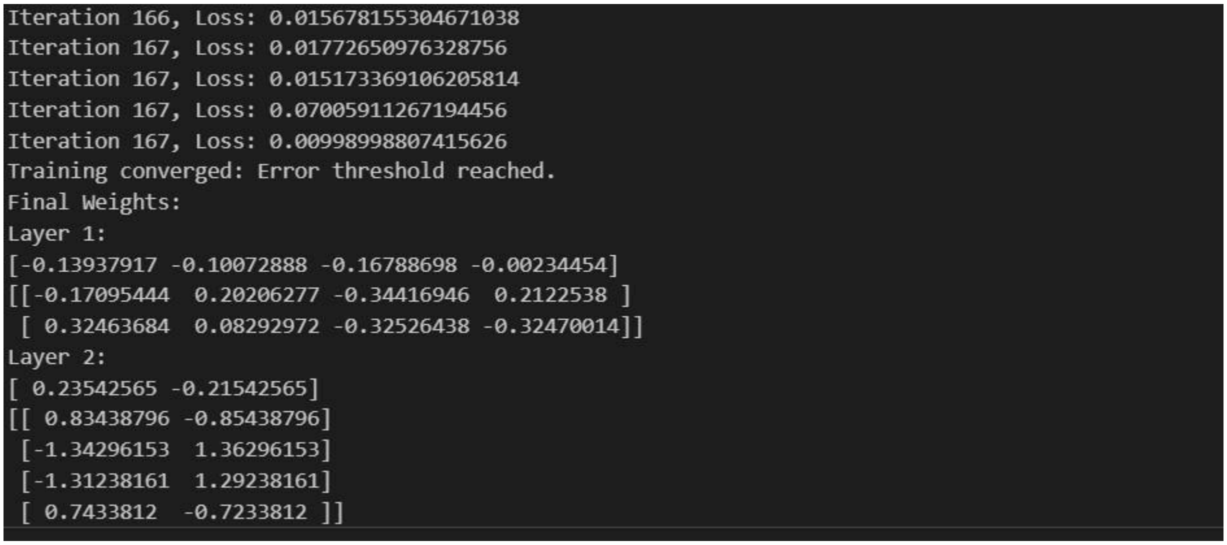
Hasil Pengujian



```
Iteration 1, Loss: 0.23640507030096652
Iteration 2, Loss: 0.23539036744879444
Iteration 3, Loss: 0.23438378905993165
Iteration 4, Loss: 0.23338532944173224
Iteration 5, Loss: 0.2323949777800034
Iteration 6, Loss: 0.2314127183172226
Iteration 7, Loss: 0.23043853053433763
Iteration 8, Loss: 0.22947238933549008
Iteration 9, Loss: 0.22851426523501334
Iteration 10, Loss: 0.2275641245460821
Training stopped by max iteration.
Final Weights:
Layer 1:
[0.23291176 0.06015346]
[[0.12884088 0.64849474]
 [0.837615   0.23158199]]
```

7. Softmax Two Layer

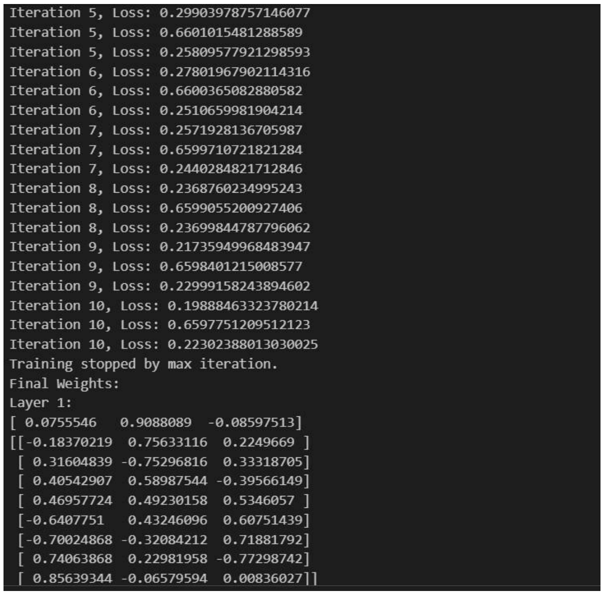
Hasil Pengujian



```
Iteration 166, Loss: 0.015678155304671038
Iteration 167, Loss: 0.01772650976328756
Iteration 167, Loss: 0.015173369106205814
Iteration 167, Loss: 0.07005911267194456
Iteration 167, Loss: 0.00998998807415626
Training converged: Error threshold reached.
Final Weights:
Layer 1:
[-0.13937917 -0.10072888 -0.16788698 -0.00234454]
[[-0.17095444  0.20206277 -0.34416946  0.2122538 ]
 [ 0.32463684  0.08292972 -0.32526438 -0.32470014]]
Layer 2:
[ 0.23542565 -0.21542565]
[[ 0.83438796 -0.85438796]
 [-1.34296153  1.36296153]
 [-1.31238161  1.29238161]
 [ 0.7433812  -0.7233812 ]]
```

8. Softmax

Hasil Pengujian



```
Iteration 5, Loss: 0.29903978757146077
Iteration 5, Loss: 0.6601015481288589
Iteration 5, Loss: 0.25809577921298593
Iteration 6, Loss: 0.27801967902114316
Iteration 6, Loss: 0.6600365082880582
Iteration 6, Loss: 0.2510659981904214
Iteration 7, Loss: 0.2571928136705987
Iteration 7, Loss: 0.6599710721821284
Iteration 7, Loss: 0.2440284821712846
Iteration 8, Loss: 0.2368760234995243
Iteration 8, Loss: 0.6599055200927406
Iteration 8, Loss: 0.23699844787796062
Iteration 9, Loss: 0.21735949968483947
Iteration 9, Loss: 0.6598401215008577
Iteration 9, Loss: 0.22999158243894602
Iteration 10, Loss: 0.19888463323780214
Iteration 10, Loss: 0.6597751209512123
Iteration 10, Loss: 0.22302388013030025
Training stopped by max iteration.
Final Weights:
Layer 1:
[ 0.0755546   0.9088089  -0.08597513]
[[-0.18370219  0.75633116  0.2249669 ]
 [ 0.31604839 -0.75296816  0.33318705]
 [ 0.40542907  0.58987544 -0.39566149]
 [ 0.46957724  0.49230158  0.5346057 ]
 [-0.6407751   0.43246096  0.60751439]
 [-0.70024868 -0.32084212  0.71881792]
 [ 0.74063868  0.22981958 -0.77298742]
 [ 0.85639344 -0.06579594  0.00836027]]
```

Pembagian Tugas

NIM	Nama	Tugas
13521010	Muhamad Salman Hakim	Membuat Laporan
13521013	Eunice Sarah Siregar	Membuat Backpropagation
13521018	Syarifa Dwi Purnamasari	Membuat Backpropagation
13521027	Agsha Athalla Nurkareem	Membuat Backpropagation